# Collections

## Definition, Editing, Browsing

**Name:** Undefined

**Contents:**

20020186691
6591094
6529723
6237114
5689706
5572709

**Comment:**

**Database:**

US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Save | Save As | Reset | Quit

Print | Search | Get Images | Classification Info

Collection Directory

L16: Entry 2 of 2 File: USPT Sep 6, 2005

DOCUMENT-IDENTIFIER: US 6941451 B2
TITLE: Management subsystem and method for discovering management device functions

Brief Summary Text (3):
This invention relates to system hardware detection and, more particularly, to the detection of system components and their features and capabilities.

Brief Summary Text (6):
However since the component and feature set map is hard-coded into a given controller, it may be difficult to interchange controllers from one system to another. In addition, it is possible that a component may be installed that either includes features not in a particular controller's feature set map or the component type is not in the controller's component map. In either case, the controller may not be able to access or control all of the component's features or may not be capable of communicating with that particular component at all.

Detailed Description Text (15):
Since system management device 110 and 115 may each have similar functions, they may use the same schemas. Thus to differentiate between functions performed by different devices, system controller 100 assigns a unique identifier (ID) to each function or feature that a given device may perform. In one embodiment, the unique ID may be a four digit hexadecimal number, although other embodiments may use other numbers of digits and other numbering systems. For example, the schema associated with "power on" for system management device 110 and 115 is 200.201. However, the unique ID for the power function of system management device 110 may be #0068h and the unique ID for the power function of system management device 115 may be #0072h, for example. Thus, once system controller 100 determines which devices are available and which functions they may perform, system controller 100 may generate and maintain a function list including a unique ID for each function of each system management device coupled to it. It is contemplated that in other embodiments, the function list may be pre-generated and stored within NV storage device 105. In such embodiments, system controller 100 may access the function list upon start up and store a copy within system controller 100.

Detailed Description Text (26):
System controller 100 may now perform functions such as monitoring system events, for example. If a request is received to send the function list (block 225), system controller 100 transmits each item in the function list which is cached within system controller 100 to the requestor (block 230). During operation, system controller may receive a request using the function list shorthand (block 235). Upon receiving a shorthand request system controller may look up the shorthand using the function list that it has cached and then translate the corresponding function into a command (block 240). For example, the command may request a temperature reading in degrees C from system management device 110. Upon receiving the temperature reading result, system controller 100 may transmit a response to the requestor (block 245), using the shorthand encoding as described above in conjunction with the description of FIG. 1.

# Freeform Search

Database: US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term: L15 and (map$ near component$)

Display: 100 Documents in Display Format: - Starting with Number 1

Generate: Hit List [✓ Hit Count] Side by Side Image

Search Clear Interrupt

## Search History

DATE: Saturday, January 28, 2006 Printable Copy Create Case

| Set Name side by side | Query | Hit Count | Set Name result set |
|---|---|---|---|
| | *DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR* | | |
| L16 | L15 and (map$ near component$) | 2 | L16 |
| L15 | L14 and availab$ | 185 | L15 |
| L14 | L13 and (system near component) | 210 | L14 |
| L13 | monitor$ near system near function | 1023 | L13 |
| L12 | monitor$ near system near function$ | 1254 | L12 |
| L11 | L10 and chang$ | 34 | L11 |
| L10 | L9 and state | 34 | L10 |
| L9 | L7 and map$ | 36 | L9 |
| L8 | L7 and msp$ | 1 | L8 |
| L7 | L6 and (system near function) | 111 | L7 |
| L6 | L4 and availab$ | 111 | L6 |
| L5 | L4 and availablitity | 0 | L5 |
| L4 | L3 and (chang$ or updat$) | 133 | L4 |
| L3 | L2 and (system near function) | 143 | L3 |
| L2 | L1 and ((monitor$ or maintain$) near (system near component$)) | 1169 | L2 |
| L1 | system near component$ | 168837 | L1 |

END OF SEARCH HISTORY

Two types of alerts may be available. The first kind of alert is a spot alert which is based on current data only. A spot alert indicates that a particular value of a system component has exceeded a threshold value. For example, a spot alert may result when the number of parity errors exceeds a predetermined threshold, or when the root partition of a disk exceeds 99%. A patch configuration problem provides another example of a spot alert. For example, assume the patch configuration problem exists for a particular patch in a particular OS release. If a host state contains the token indicating the presence of the particular patch as well as the token indicating the particular OS release, an alert is issued.

Detailed Description Text (38):
The second type of alert is a predictive alert. A predictive alert analyzes historical and current data to identify trends. In other words, the predictive alert is a form of trend analysis. Storing multiple instances of stored host states in the host state data base, makes possible such trend analysis of the operating conditions of a monitored system. Trend analysis allows pro-active detection of undesirable conditions in the collected diagnostic data. For example, trend analysis identifies that the number of memory parity errors is increasing, even though the number is not yet fatal. The alert can generate the probability that the increase will eventually result in a fatal error. Another example of a predictive alert is memory leak detection.

Detailed Description Text (39):
Trend analysis compares the value of a current alert to previous alert results. The trend is determined by comparing, e.g., tokens containing the number of parity errors of a memory element, over a sequence of host states. Trend analysis may use alerts saved from a previous analysis or may obtain relevant token values from saved host states or may operate on both saved tokens from earlier host states as well as saved alert values.

Detailed Description Text (42):
As an example of a predictive alert consider an alert that predicts whether or not swap space is going to get low on the system. The token value used is one that identifies swap-space used. An operator that is useful in predictive analysis is one called, OverTimeOperator, that provides the value of swap spaced used over time, i.e., from sequential host states. One can specify how far back the OverTimeOperator should go in retrieving token values from previous host states. The spot test of such a token determines if in the latest data, the swap spaced used is over 90%. That is the first gating factor of the alert. Then the alert uses that spot test data and the data from the OverTimeOperator and provides the data to a normalization function which provides a graphical analysis of the data. If the angle of normalization is greater than 52 degrees, an alert is generated thereby predicting that swap space is going to get low on the system. The particular angle selected as a trigger may depend on such factors as the system being monitored and the normalization function.

Detailed Description Text (51):
In one embodiment, the alert definitions are run against the host states using alert functions. The code for each alert definition is not actually stored in the Alert function. Instead, the Java language code for the alert definition is sent by the alert editor to a file repository, e.g., alert types 243 from the compiler. A reference to the compiled alert definition is then stored in the Alert Function which is stored in a database, e.g. database 109 as shown in FIG. 1. An exemplary AlertFunction class is shown below.

Detailed Description Text (52):
Thus, an Alertfunction object will exist for each alert definition, the object pointing to the location where the alert definition actually is stored. The Alertfunction object will be run against the host state (or states) as appropriate.

Detailed Description Text (53):
In one embodiment, there are five possible output levels of severity, red, yellow, blue, black, green. Weight creates a range mapping onto some or all of these severitys. For instance, if a particular alert returns a number between 1 and 100, a level of between 1 and 20 may be mapped onto red. Similarly, for an alert that returns a value of true or false, a true value can be mapped onto, e.g., red. For each new host state, the alert processor retrieves all of the alert

functions. Each alert function points to the associated compiled alert code and in this way all of the alert definitions are parsed against the host state.

Detailed Description Text (54):
When alerts are created, that is when the alert definitions pointed to by the alert functions, are found to exist in a particular host state(s), then an alert object in accordance with an alert class is created. An exemplary alert class is as follows:

Detailed Description Text (57):
In a preferred embodiment, alert definitions are processed on each host state after it is generated. Each alert type is compared to a host state and an output is generated. That is, the tokens contained in the host state are compared to the condition defined in the alert type. An alert editor 221 allows alert types to be defined through an editor. An alert, which is an instantiation of a particular alert type, can have an associated severity level as previously described.

Detailed Description Text (59):
The alert types are related to the element hierarchy. The alert type to test the disk capacity of a partition, as described previously, utilizes tokens related to the partition element in the element hierarchy. That alert works fine for all partitions. In accordance with the model discussed in the element and element hierarchy, only one alert would exist for all partitions created, so all partitions that exist on all disks would have the alert processed when a host state is created.

Detailed Description Text (61):
There are various operators which are utilized to define the alerts. The operators are in the general sense functions that operate on the token types contained in the host states. Exemplary operators include logical operators, AND, OR, NOT, XOR, BIT-AND, BIT-OR, BIT-NOT, BIT-XOR, arithmetic operators, SUM SUBTRACT, MULTIPLY, DIVIDE, relational operators, LESS THAN, LESS THAN OR EQUAL, GREATER THAN, GREATER THAN OR EQUAL, EQUALS, NOT EQUALS. There are also set operators, UNION, INTERSECTION, ELEMENT OF, (element of is checking if the particular value is an element of a set), DIFFERENCE BETWEEN 2 SETS. String operators include, STRING LENGTH, STRING-SUBSTRING (to see if the string you have is actually a substring of the original string), STRING-TOKEN, (to see if this particular string is a token of the bigger string). Conversion operators convert, HEXADECIMAL TO DECIMAL, HEXADECIMAL TO OCTAL, HEXADECIMAL TO BINARY. Additional operators are, AVERAGE, MEAN, STANDARD DEVIATION, PERCENTAGE CHANGE, SLOPE (which is based on graphing a straight line interpolation of plots), SECOND ORDER SLOPE, CURVE EXPONENT (map an exponent algorithm on the actual curve), MAX, and MIN, for the maximum and minimum value, ALL OF TYPE (extracts all the values of a certain type out of a host state), ALL OVER TIME (obtains a range of data for a token over a period of time), EXIST, (checks to see if token exists), WEIGHT, (applies a certain weight to a value), NORMALIZE. Some embodiments may also provide for custom operators. Other operators may be utilized in addition to or in place of those described above.

Detailed Description Text (62):
Once the alerts have been defined and stored in alert types database 243, the alerts have to be run against the host states. Whenever a host state is created the alert and trend analysis is run against the host state. Thus, the alert types and a host state are provided to analyzer 223. The analyzer processes the alerts by running the code definition of the alerts against the host state(s). The alert types may be associated with particular elements so that an entire tree structure does not have to be searched for each alert type. If an alert is generated, alerts data base 239 stores the value of the alert. Storing the alerts in a database allows for later retrieval.

Detailed Description Text (64):
In one embodiment of the invention, all alert types are global in that the alert types are run against all monitored systems, i.e., the host state representation of that system, in a default mode. However, the tests can be can be selectively enabled (or disabled) according to the monitored system. Such capability is provided in the embodiment shown in customer alert configurer 231 which, in a preferred embodiment, is a Java programming environment based

graphical user interface (GUI) which provides the ability to select which alerts should run on particular monitored systems from a list of all the alerts available. Note that it is not essential that each system being monitored have the alerts match their actual hardware and software configuration. If an alert has no input the alert will be marked as invalid. Consider, for example, a disk mirroring alert. If the host state does not show that any disk mirroring exists on the host, then the disk mirroring alert is invalid and ignored by the system. Thus, alerts that reference elements or token parameters not found in a particular host state are marked as invalid and ignored

Detailed Description Text (72):
Once the alert types have been run against a particular host state, an alert indicating a serious problem may result. The severity of a particular alert can be determined when the alert is defined. For example, in one embodiment, an alert, which is an instantiation of a particular alert type, has output severity levels of red, yellow, blue, black, green, with red being the most serious.

Detailed Description Text (73):
For alerts which are determined to be serious, e.g., for red alerts, it is typical to want to correct the problem causing the red alert as expeditiously as possible. Thus, when the monitoring system is expected to provide support for its customers, and particularly 24 hour support, the more information regarding the problem and its solution that can be provided to the responsible engineer at any particular time of the day, the more likely that that engineer can find a resolution to the problem. This can be seen to be particularly important when the alert is a red alert since resolving the problem may avoid or minimize a critical failure in the monitored system. A monitoring computer system according to one embodiment of the present invention, therefore provides the engineer a variety of available information to assist the engineer in determining a solution to whatever problem has arisen.

Detailed Description Text (74):
In order to assist the support engineer, known problems and resolutions are stored in a systems resolution database which can be in database storage 123 (FIG. 1). For example, the systems resolution data base might include such information that a failure associated with a specific disk drive in a particular computer system can be fixed by a software patch. Thus, the data base would include an entry for that specific disk drive along with the software patch as well as a textual description of the problem and its solution. Data base 123 typically is a relational data base. For particularly serious alerts or red alerts a cross referencing system to the systems resolution data base is used to increase the information available to the support engineer.

Detailed Description Text (75):
The systems resolution data base has "system resolution" type documents prepared by engineering support staff to help solve common problems. The documents include a textual description of the problem and likely resolutions. In other words, the data base contains a description of what to check for in terms of possible causes of the problem in order to try to resolve the problem. When alerts are created the alert editor 221 can include a special function such that every time an alert is generated, a part of the alert editor prompts the creator of the alert to provide a textual problem description and potential solutions if the written description of the problem is not available from the existing systems resolution data base. Note that a solution may be identified before or after a particular alert is created.

Detailed Description Text (78):
Referring to FIG. 13, a system according to the present invention functions as follows. Alerts are run against the incoming host state in 1301. When an alert condition is detected, if there is a severe alert detected in 1303, textual information in the alert, for instance, keywords such as "swap spaced used" is auto-cross-referenced against the systems resolution database of known problems and resolutions in step 1305. In addition, information relating to known problems and resolutions may be cross-referenced against the known history and status of the system being monitored. Known history and status may be obtained by extracting relevant information from prior host states in 1307. Historical information for the system being monitored allows the support engineer to know if the problem is a repeat problem.

Detailed Description Text (79):
Where necessary, a parent of a particular token is identified as explained more fully herein. The relevant information from the current host state is obtained in step 1309. The information from the systems resolution data base and the related host state information, both current and historical, are combined and presented to the support engineer in 1311. The state of the rest of the system is then investigated and the customer is provided a notification of the problem and the solution in step 1313. This can be accomplished by logging a service order into a call management system, which contacts an administrator of the monitored system, e.g., via the email address contained in the profile data base. The administrator is provided a description of the problem and the appropriate solution based on the textual information and the solution contained in the systems resolution data base and the analysis by the support engineer.

Detailed Description Text (81):
An exemplary red alert processing according to the present invention is described in the following example. Assume that a monitored system, which has a number of hard disks, is not seeing one of its disks for some reason. Assume also that an alert exists that effectively monitors the disks that are attached to the machine and if one disk goes missing, the alert recognizes that fact. That can be detected, for example, by comparing the present disks detected to the disks detected in a previous host state, i.e. a previous time slice. The problem with the disk may originate from a number of sources. For example, it may be that somebody has unplugged it, or more likely, the disk controller is faulty or a disk is completely dead. Assume also that the alert which "fires" is a red alert. A process in alert processing 247 retrieves textual information related to problem resolution that was provided by the creator of the red alert or that came from the problem resolution data base. The systems resolution data base 123 is searched based on parameters of the red alert, such as the name of the alert, to see if there are any entries in the systems resolution data base that are related to the problem identified by the red alert. In addition, the alert processing 247 then retrieves information from the host state that is related to the problem based on information in the systems resolution data base 123. Related information includes information that is related to the problem in the sense that it is related on the tree. For example, referring to FIG. 5, information related to fixed media device 511, e.g., a hard disk, may include the peripheral adapter 503, peripheral bus 309 and host 301.

Detailed Description Text (82):
The way the host state data structure is held in a preferred embodiment, is in regular expression and thus is searchable. If a token is missing such as would be the case if the disk is missing, then step 1307 (FIG. 13) can find the missing token by going back in time (to a prior host state) when the disk was there and the parent of that disk can be determined. Then the processing can return to the current host state and search backwards from the parent. Because the tree structure is an inverted tree type structure, the tree can be parsed back up by following the links returning each particular piece of information about each element in the tree all the way to the top. The related information is returned because there is a possibility that if something has gone wrong with the disk, there might be something wrong with the controller or the bus coupling the disk controller to the host and so relevant information is extracted from the tree and provided to the support engineer. Such information may be required to follow the textual description for trouble shooting the particular problem.

Detailed Description Text (83):
Therefore it is possible to traverse up the tree from the spot where the missing disk was supposed to be located. Where the disk failed for another reason, e.g., the number of soft errors exceeded a threshold, rather than going missing, it is not necessary to use previous host states to determine the appropriate location in the tree from which to traverse back towards the root. Once the information from the tree is retrieved, that information, along with the textual information from the systems resolution data base can be combined, e.g., into a text file and provided to the support engineer. In addition, the information containing the textual information and the tree information can be logged into a data base used by support engineers to track problems.

Detailed Description Paragraph Table (1):

TABLE 1 Class Test Name Description network automount.files Automount/etc Files automount.nis+ Automount NIS+Files automount.nis Automount NIS Files dfshares NFS shared filesystems domainname Domain name etc.defaultdomain /etc/defaultdomain etc.defaultrouter /etc/defaultrouter etc.dfstab List/etc/dfs/dfstab etc.hostnames /etc/hostname (s) etc.hosts /etc/hosts etc.mnttab List/etc/mnttab etc.named.boot /etc/named.boot etc.nsswitch.conf /etc/nsswitch.conf etc.resolv.conf /etc/resolv.conf netstat-an List aIl TCP connections netstat-in List network interfaces netstat-k Network interface low-level statistics netstat-rn List network routing nisdefaults NIS+ server defaults nisstat NIS+ statistics ypwhich NIS server name ypwhich-m NIS map information OS checkcore Check for core files df Disk Usage dinesg Boot Messages framebuffer Default console/framebuffer hostid Numeric ID of host ifconfig Ethernet/IP configuration messages System messages (/var/adm/messages) patches List system patches pkginfo Software package information prtconf System hardware configuration (Software Nodes) prtconf-p System hardware configuration (PROM Nodes) prtdiag Print diagnostics (Sun-4d systems only) sar System activity reporter share Shared directories showrev Machine and software revision information swap Swap report uptime Local uptime and load average whatami Lengthy system description report unbundled fddi-nf_stat FDDI low-level statistics metastat Online DiskSuite or Solstice DiskSuite vxprint Systems using SPARCstorage Array Volume Manager x25_stat X.25 low-level statistics

CLAIMS:

1. A method of monitoring a monitored computer system, comprising:

determining if a condition exists in current state information stored in a first storage location in a monitoring computer system, the current state information indicating a state of hardware and software components and operating conditions of the monitored computer system during a first time period, and wherein the current state information is represented as a tree structure, the tree structure including component information which represents the hardware and software components and operating conditions of the monitored computer system, the component information being extracted from diagnostic data provided by the monitored computer system;

retrieving component information related to the condition from the current state information when the condition is determined to exist;

presenting the component information on a display device;

providing a third storage location storing previous state information relating to the monitored computer system, the previous state information being represented as another tree structure, representing a state of the monitored computer system during a second time period, the second time period being different from the first time period; and

traversing the other tree structure to identify a missing component in the tree structure of the current state information and providing an indication of the missing component.

4. The method as recited in claim 1 further comprising retrieving second component information related to the condition from the previous state information.

6. The method as recited in claim 1 further comprising traversing the tree structure of the current state information according to the indication of the missing component in order to extract the component information related to the condition from the tree structure of the current state information.

Previous Doc Next Doc Go to Doc#

L11: Entry 24 of 34 File: USPT May 22, 2001

DOCUMENT-IDENTIFIER: US 6237114 B1
TITLE: System and method for evaluating monitored computer systems

Abstract Text (1):
A computer system used in monitoring another computer system provides both textual resolution information describing a likely solution for a problem encountered in the monitored computer system as well as component information that relates to the particular problem. The component information includes the various hardware, software and operating conditions found in the monitored computer system. The monitoring computer system determines if a condition of a predetermined severity exists in the monitored computer system according to diagnostic information provided from the monitored computer system. The diagnostic information is represented in the monitoring computer system as a hierarchical representation of the monitored computer system. The hierarchical representation provides present state information indicating the state of hardware and software components and operating conditions of the monitored computer system. The resolution information relating to the condition is retrieved from a resolution database and relevant component information is retrieved from the hierarchical representation of the computer system and presented to a support engineer to assist them in diagnosing the problem in the monitored computer system.

Parent Case Text (2):
This application relates to the following commonly owned co-pending applications, Ser. No. 08/819,500, entitled "DYNAMIC TEST UPDATE IN A REMOTE COMPUTER MONITORING SYSTEM", by Michael J. Wookey, filed Mar. 17, 1997; Ser. No. 08/819,501, now U.S. Pat. No. 6,023,507, entitled "AUTOMATIC REMOTE COMPUTER MONITORING SYSTEM", by Michael J. Wookey, filed Mar. 17, 1997; Ser. No. 08/829,276, entitled "REBUILDING COMPUTER STATES REMOTELY", by Michael J. Wookey; Ser. No. 08/854,788, entitled "REMOTE ALERT MONITORING AND TREND ANALYSIS", by Michael J. Wookey et al., filed May 12, 1997; Ser. No. 08/861,141, "AUTOMATIC BUILDING AND DISTRIBUTION OF ALERTS IN A REMOTE MONITORING SYSTEM", by Michael J. Wookey et al., filed May 21, 1997, which applications are incorporated herein by reference.

Brief Summary Text (6):
Computer systems are typically serviced when a failure is noticed either by system diagnostics or by users of the system when the system becomes partially or completely inoperative. Since computer systems are frequently located at some distance from the support engineers, when problems do occur, a support engineer may access the computer system remotely through a modem in an interactive manner to evaluate the state of the computer system. That remote dial-in approach does allow the support engineer to provide assistance to a remote customer without the delay of traveling to the computer system site. Once connected to the remote computer system, the support engineer can perform such tasks as analyzing hardware and software faults by checking patch status, analyzing messages file, checking configurations of add-on hardware, unbundled software, and networking products, uploading patches to the customer system in emergency situations, helping with problematic installs of additional software, running on-line diagnostics to help analyze hardware failures, and copying files to or from the customer system as needed.

Brief Summary Text (11):
In one embodiment, the invention provides a method of monitoring a computer system, which includes determining if a condition of a predetermined severity exists in the monitored computer system according to diagnostic information provided from the monitored computer system. The diagnostic information is included in a hierarchical representation of the

monitored computer system. The hierarchical representation provides present state information indicating the state of hardware and software components and operating conditions of the monitored computer system. The resolution information relating to the condition is retrieved from a first storage location and component information from the hierarchical representation of the computer system relating to the condition is retrieved from a second storage location.

Brief Summary Text (12):
In another embodiment, the invention provides a computer system for monitoring a monitored computer system. The computer system includes a first storage location storing at least a first host state representing a state of the monitored computer system during a first time period. The first host state is in a tree structure that includes component information which represents hardware and software components and operating conditions of the monitored computer system. The component information is extracted from diagnostic data provided from the monitored computer system. A second storage location stores information providing a description of known problems and resolutions. A third storage location stores a first program code segment which is run against the host state to identify a predetermined condition in the host state. A fourth storage location stores a second program code segment which retrieves resolution information from the second storage location relating to the predetermined condition and retrieves component information relating to the predetermined condition.

Drawing Description Text (3):
FIG. 1a shows an exemplary system for rebuilding the state of a computer system.

Drawing Description Text (5):
FIG. 2 details the architecture of a system that rebuilds computer states.

Drawing Description Text (18):
FIG. 11 shows another example of a host state.

Drawing Description Text (19):
FIG. 12 shows how the host state can be displayed to show graphical, and attribute information about the host state.

Detailed Description Text (4):
Referring to FIG. 2, the architecture of a system according to one embodiment of the present invention, is shown in greater detail. Incoming diagnostic data 201 is received via email or direct modem link (or another communication link) into the monitoring system and stored in raw test data storage area 213. The test data, which contains information about the software and hardware components in monitored system 102, is processed by token processing 211 to extract the information associated with hardware and software components in the monitored system. The extracted information is then used to create a representation of the monitored system in host state creator 206 based on the component information. The host state is the state of the monitored system or one computer of the monitored system over the particular time period that the diagnostic tests were run. The host state is described further herein.

Detailed Description Text (5):
In order to create a representation of the monitored system, the components contained in the test data are rebuilt into a system hierarchy based on a static hierarchy tree definition. In a preferred embodiment, one static hierarchy tree definition is applicable to all systems which are being monitored. The extracted information about the components in the monitored system are mapped onto the static tree to create the system representation for the monitored system. Thus, the state of the monitored system is rebuilt.

Detailed Description Text (6):
The hierarchy tree is composed of elements. An element can be thought of as a physical or virtual component of a computer system. For example, a computer system may include such components as a disk, a disk partition, a software package, and a software patch. An element has tokens associated with it. Thus, a disk partition element may have a disk percentage token, disk name token, and space available token associated with it. An element definition includes what token types fulfill the element, and give the element value. In one embodiment, an element

is an instance of a class of element types as implemented in an object oriented language such as the Java.TM. programming language. Java and Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Detailed Description Text (7):
An exemplary portion of a static tree definition a computer system is shown in FIGS. 3-6. FIG. 3 shows lower level (closer to the root) elements of the static tree and FIGS. 4, 5 and 6 show how the tree definition expands. The element host 301 defines the kind of computer that is being monitored. For instance, the host may be a workstation running an operating system such as Solaris.TM. (which is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries) or a personal computer running another operating system such as WINDOWS NT (WINDOWS NT is a trademark of Microsoft Corporation). Attached to host 301 are other physical or virtual components such as central processing unit (CPU) bus 303, monitor 305, keyboard/mouse 307, peripheral bus 309 and software configuration 311. Note that the terms are very general. Each element represents types of components that can be found in a typical computer system.

Detailed Description Text (16):
There are two types of tokens. The first is an element realizing token. Element realizing tokens provide a way to determine whether an element should be included when building a particular host state. For example, a disk name token is an element realizing token. The second type of token are data tokens which provide additional information about an element that has already been realized, such as the token indicating the number of sector per cylinder. Thus, it can be seen that tokens give value to the elements.

Detailed Description Text (17):
For any particular system, it is preferable to create tokens with as much granularity as possible. Thus, the smallest piece of information that is available about a system from the available diagnostic tests should be included as a token. Representative tokens are included in the description herein. The exact nature of the tokens and the total number of tokens will depend upon the system that is being monitored, including its hardware and operating system, and the diagnostic tests that can be run on the system. An exemplary output of one the diagnostic tests is shown in FIG. 8. The processing must extract from the output such information as the disk partition ID, last sector, first sector and the like.

Detailed Description Text (18):
Further example of elements, tokens and associated test are found in application Ser. No. 08/829,276, entitled "REBUILDING COMPUTER STATES REMOTELY", previously incorporated herein by reference.

Detailed Description Text (21):
Once all the raw test data has been processed and completed token data in available in token data base 207 is available, the second set of processing operations to build the representation of the monitored computer may be completed. In order to understand the building of the tree, an examination of several typical features of an element class will provide insight into how an element is used to build a tree.

Detailed Description Text (24):
Referring to FIG. 9, an example of building a host state based on the elements of the static tree is shown. The term "host state" refers to the representation of the monitored system based on its diagnostic data. The host state essentially describes the state of a system for a given time period. The host state may be viewed as an instantiated element hierarchy based on the raw data that has come in from the remote host. In other words, it is a completed element hierarchy with value. The diagnostic data is collected over a particular time period, so the host state represents the state of the monitored machine over that particular time period, e.g., an hour. The host state is built by starting from the top of the tree element host 301 (shown in FIG. 3). The element 301 has Get Host method 901 to retrieve relevant tokens from the token data base 207. As shown in FIG. 9, the element 301 is realized as "labtis 7" 903. Because the token data base is a hashtable in the preferred embodiment, the realization of each element is faster. Next element graphics adapter 501 (FIG. 5) gets graphics adapter designated as

cgsix0914 and ffb0916 using Get graphics adapter method 911. Continuing to build the host state, media controller element gets SCSI0912 from token data base 207. In a preferred embodiment, the host state is built in depth order meaning that each element and all branches of that element are built before another element is built. Thus, referring back to FIG. 5, for example, everything on peripheral bus 309 is built before the building of the software configuration 311. For each element in the static tree, the token data base 207 is searched and the host state is created in element fulfillment processing 205 which requests tokens from token data base 207 in the form of searches for tokens providing realization and value to the static tree.

Detailed Description Text (25):
Once the element fulfillment stage is completed a final token post processing operation takes place in 208. An element can have a token defined that is the mathematical result of other tokens. For example, a disk space free token is derived from a simple subtraction from a disk used token and a total disk space token. The calculations are completed in this post processing operation 208 to complete the host state.

Detailed Description Text (26):
Note that because the tree definition is static and is intended to be general, not all elements will be found in every host state. Thus, when building the host state, no data will be found in the token data base for a particular element that is lacking in the monitored system. Additionally, in some host states, an element will be found more than once. Thus, the tree structure provides the flexibility to build host states that look very different.

Detailed Description Text (27):
Once the host state is built, it is saved in host states storage 209. The storage of the host state provides several advantages. For one, it provides the capability to search back through time and to compare one host state with another host state from a different time or perform trend analysis over time. The host states may be stored for any amount of time for which adequate storage area is available. For example, host states may be stored for a year.

Detailed Description Text (28):
Additionally, the stored host states are used when the diagnostic data is incomplete. There may be occasions when a test has failed to run in the monitored system or has not run before a scheduled communication of data from the monitored system. That may cause problems in the building of the host state from the static tree, especially where the test was one that created elements lower in the tree (i.e. towards the root). Each element can include a value that indicates how critical the element is to the system. If the element is critical, such as a disk, there may be a problem with the system and it should be noticed. If the data is not critical to the system, then older data may be retrieved from the previous host state in time for that particular host. That may be limited by restricting such retrieval to a specified number of times, e.g., 10, or any other number appropriate to the criticality of the element, before marking data as invalid.

Detailed Description Text (30):
A second example of building a host state is shown in FIG. 10. Element 1001 has associated token types for the name of the system and the operating system. Peripheral bus element 1003 has associated token types which gets the name of the peripheral bus and any onboard RAM. Element 1005, which is a processor element, has associated token types to provide a name, a revision number and the processor speed. The static definition 1000 creates a host state 1020 where the system is realized as "Spike" with an OS release of 5.4. The peripheral bus is instantiated as SBUS0 with 512 K of RAM. The processor element is instantiated three times as MPU01006, MPU11008 and MPU21010. Thus, an example is provided where a single element is realized more than one time in a particular system.

Detailed Description Text (31):
Referring to FIG. 11, another example of a host state is provided. The system is shown as element 1101 with associated values of being SPARCstation7, with a system name Spike and an OS 5.4 release. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc., in the United States and other countries. Products

bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. The system has a peripheral bus, Sbus0, which has two SCSI buses 1105 and 1107. Attached on SCSI bus 0 are two disks sd0 and sd1. Disk "sd0" has associated tokens, in addition to its name, the manufacturer 1113, the revision 1115, the size of the disk, 1117 and the serial number 1119.

Detailed Description Text (32):
In addition to storing the host state in data base 209, the system provides a graphical interface to access information about the host state. Referring to FIG. 12, an exemplary system visualization screen is shown. The tree structure is provided in region 1201 of the screen which graphically represents a portion of the host state shown in FIG. 11. Tree structures may also be represented in the form shown in FIGS. 7a-7e or other appropriate form. In addition to displaying the tree structure which provides the user a graphical depiction of the completed element hierarchy for a particular system at a particular time, the screen also provides a graphical image of the particular component which is being viewed. For instance, region 1203 of the screen shows a graphical image 1205 of a disk. Assuming that the viewer had clicked on disk 1202, sd0, region 1207 shows the attributes or token values associated with the selected element. Thus, the attributes relating to name, manufacturer, revision, size and serial number are all provided. This presents the support engineer with an easily understandable graphical image of the total system, and any particular component of the system that is represented in the host state, along with pertinent attributes.

Detailed Description Text (33):
Referring again to FIG. 2, the system visualizer 225 receives host states from host states database 209 and customer system information stored in data base 235. The system visualizer also receives alerts and local configurations relevant to a particular support engineer. One task of the system visualizer is to select the particular host that is to be worked upon or viewed. Thus, the system visualizer searches the host states database 209. The visualizer provides the ability to parse through time to select from all the host states available for a particular system. While each element may have a graphic associated with it, a separate graphic can be used to indicate that a problem exists with a particular element.

Detailed Description Text (34):
In addition to displaying the attributes of an element, which are the values of the tokens associated with the element, the system visualizer provides graphical capability to graph attributes against time. One or more attributes can be selected to be graphed against history. In other words, the same attributes from different instances of the element hierarchy for a particular system can be compared graphically. For example, the amount of disk free over time can be monitored by looking at outputs of the "df" test over a period of time. The df output includes such token values as disk percentage used for a particular partition, partition name and size of partition. The visualizer will extract the tokens representing amount of disk percentage used for a particular set of host states. The host states from which the disk percentage tokens are extracted is determined according to the time period to be viewed. That information can then be visualized by plotting a graph of disk percentage used against time. Also, the visualizer can view different instances of the host state. In other words, the visualizer can view the state of a monitored system at different times. That capability provides a visual interpretation of changes in system configuration. The visualizer accesses the stored multiple instances of the host state of the particular system to provide that capability.

Detailed Description Text (36):
Once host states have been created, the data can be analyzed for the presence of alerts. Alerts are predefined conditions in the various components of the monitored computer system that indicate operating conditions within the system. The alerts are designed to be sufficiently flexible so that they can detect not only serious problems, but also detect performance and misconfiguration problems. Different levels of severity may be provided in each alert. For example, alert severity can range from one to six. Severity level six indicates effectively that the system has gone down while a severity level of one indicates that there may be a performance problem in the system.

Detailed Description Text (37):